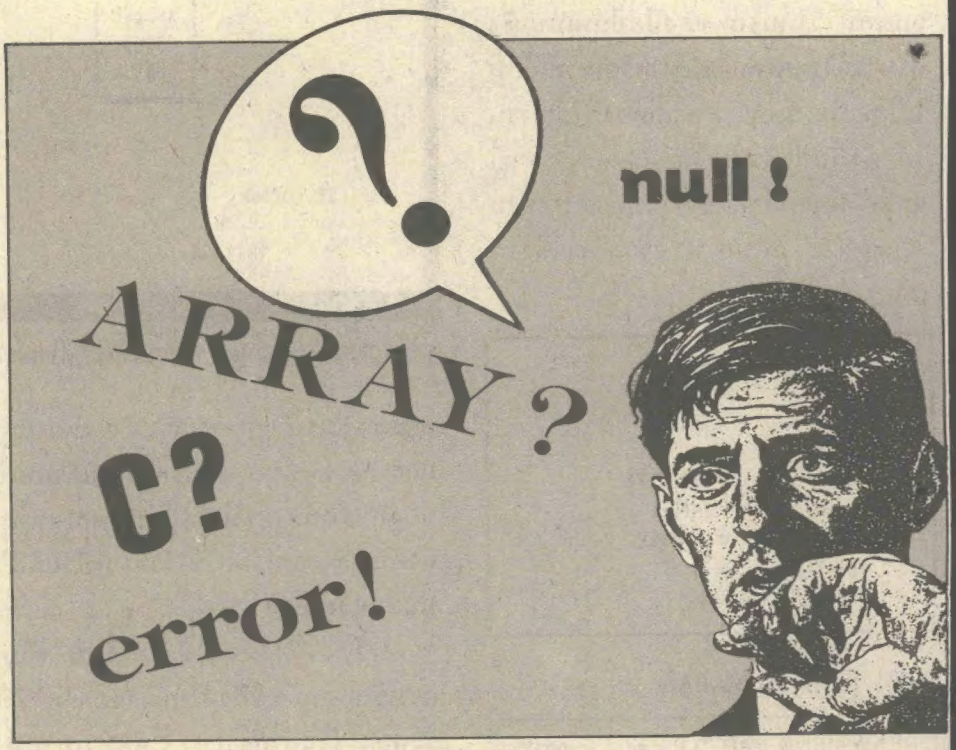
สรรค์ เสกขุนทด

กระเทาะเปลือก ภาษาซี ตอนจบ

กล่าวถึงเรื่องราวภาษาซีอีกครั้ง กับการถามและตอบปัญหา เพื่อไขข้อข้องใจบางประการ



คำถาม 14 ข้อที่น่าสนใจก็ได้ผ่านสายตาท่านผู้อ่านไปแล้ว ซึ่งในตอนที่ 1 ส่วนมากจะกล่าวถึงปัญหาเกี่ยวกับเรื่อง null ในตอนจบนี้จะขอกล่าวถึงปัญหาที่เหลือทั้งหมด เช่น Arrays, Pointer, ANSI C เป็นต้น

1. ในไฟล์ภาษาซี ไฟล์หนึ่งมีการประกาศดังนี้

```
char a[5];
```

อีกไฟล์หนึ่งประกาศดังนี้

```
extern char *a
```

ทำไมถึงเกิด error ได้

ตอบ เพราะชนิดข้อมูลของ a ไม่ตรงกัน ชนิดข้อมูล "พอยน์เตอร์ไปยังชนิดข้อมูล T" เป็นคนละอย่างกับ "อะเรย์ของชนิดข้อมูลของ T"

*2. แต่มีบางคนบอกว่า char a[ ] เหมือนกับ char *a อย่างนี้หมายความว่าอย่างไร*

ตอบ ความเหมือนที่ว่านี้ (ความเหมือนนี้หมายความว่าสามารถประกาศในรูปพอยน์เตอร์หรืออะเรย์ก็ได้) มีผลเฉพาะตอนประกาศพารามิเตอร์ของฟังก์ชัน (format parameter) เท่านั้น ความเหมือนนี้มีผลจากการที่อะเรย์จะเปลี่ยนสภาพ (decay) เป็นพอยน์เตอร์เมื่ออะเรย์อยู่ในนิพจน์ นั่นคือเมื่อจะมีการใช้อะเรย์ในนิพจน์ใด ๆ อะเรย์นั้นจะถูกแปลงให้เป็นพอยน์เตอร์ไปยังสมาชิกแรกของอะเรย์แทน ไม่มีการส่งผ่านสมาชิกอะเรย์ไป เมื่อเป็นเช่นนี้ฟังก์ชันที่รับค่าจะต้องประกาศพารามิเตอร์ในรูปของพอยน์เตอร์ไม่ใช่อะเรย์ แต่ในกรณีนี้ตัวภาษายอมให้ประกาศในรูปของอะเรย์ได้ เพื่อให้ดูราวกับว่ามีการผ่านสมาชิกอะเรย์ไป ซึ่งทำให้อ่านโปรแกรมได้ง่าย โปรแกรมเมอร์บางคนชอบแบบนี้ เพราะเป็นเครื่องย้ำว่าตอนนี้กำลังจัดการกับอะเรย์ (ไม่ใช่พอยน์เตอร์ของ object อันเดียว)

เนื่องจากฟังก์ชันจะไม่ได้รับพารามิเตอร์ที่เป็นอะเรย์ ดังนั้นการประกาศในลักษณะนี้

```
f(a)
char a[ ];
```

พารามิเตอร์ a จะมีชนิดข้อมูลเป็น "พอยน์เตอร์ไปยัง char" ไม่ใช่ "อะเรย์ของ char" ซึ่งมีค่าเท่ากับการประกาศดังนี้

```
f(a)
char *a
```

ดังนั้นอะเรย์กับพอยน์เตอร์จึงเหมือนกันในลักษณะดังกล่าวโปรดจำไว้ว่าความเหมือนนี้ใช้ได้เฉพาะตอนประกาศพารามิเตอร์ของฟังก์ชันเท่านั้น ในกรณีของการใช้ในรูปของอะเรย์นั้น มีคนจำนวนมากเชื่อว่า ก่อให้เกิดความสับสนมากกว่าจะได้ประโยชน์ อย่างไรก็ตามการจะเลือกใช้วิธีใดขึ้นกับความพอใจเป็นสำคัญ

3. ที่ว่า "พอยน์เตอร์กับอะเรย์สมมูลกัน (equivalence) นั้นเป็นอย่างไร

ตอบ ความสับสนเรื่องพอยน์เตอร์ในภาษาซี ส่วนใหญ่จะเกิดจากเรื่องความสมมูลของพอยน์เตอร์และอะเรย์นี้เอง "ความสมมูล" ไม่ได้หมายความว่าพอยน์เตอร์และอะเรย์ใช้แทนกันได้แต่หมายถึงว่า

– อะเรย์จะเปลี่ยนสภาพเป็นพอยน์เตอร์เมื่ออยู่ในนิพจน์

– อะเรย์และพอยน์เตอร์สามารถอ้างถึงข้อมูลได้โดยใช้ซับสคริบ (subscript) เช่น

```
Char a [10];
Char *p = a;
int i;
```

เราสามารถอ้าง a[i] และ p[i] ได้

4. เหตุใดการผ่านอะเรย์ 2 มิติ ไปให้ฟังก์ชันที่รับพารามิเตอร์ชนิดพอยน์เตอร์ไปยังพอยน์เตอร์จึงเกิด error

ตอบ กฎที่ว่าอะเรย์จะกลายเป็นพอยน์เตอร์ไม่ได้มีลักษณะแบบ recursive นั่นคือ "อะเรย์ของอะเรย์" จะกลายเป็น "พอยน์เตอร์ไปยังอะเรย์" ไม่ใช่ "พอยน์เตอร์ไปยังพอยน์เตอร์" (พอยน์เตอร์ไปยังอะเรย์ สร้างความสับสนได้ง่าย ควรหลีกเลี่ยงการใช้) ถ้าจะผ่านอะเรย์ 2 มิติไปยังฟังก์ชันเช่น

```
int array [YSIZE] [XSIZE];
f(array);
```

ฟังก์ชัน f ต้องประกาศพารามิเตอร์ดังนี้

```
f (int a[  ] [XSIZE]) {...}
หรือ f (int (*ap) [XSIZE]) {...}
```

การประกาศอันแรก คอมพายเลอร์จะทำหน้าที่แปลง "อะเรย์ของอะเรย์" ให้เป็น "พอยน์เตอร์ไปยังอะเรย์" ส่วนอันที่สอง เป็นประกาศ "พอยน์เตอร์ไปยังอะเรย์" โดยตรง ทั้งสองกรณีต้องบอกขนาดของคอลัมน์ (XSIZE) แต่ไม่จำเป็นต้องบอก ROW (YSIZE) ดังนั้นถ้าฟังก์ชันรับพารามิเตอร์แบบพอยน์เตอร์ไปยังพอยน์เตอร์และส่งอะเรย์สองมิติไป จึงเกิด error

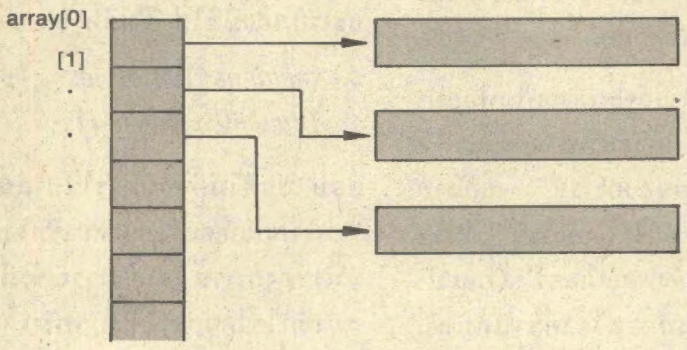
5. "พอยน์เตอร์ไปยังอะเรย์" ใช้ในกรณีไหนและประกาศอย่างไร

ตอบ ปกติไม่ค่อยมีการใช้มากนัก พิจารณาพอยน์เตอร์ไปยังสมาชิกใดสมาชิกหนึ่งของอะเรย์ เราสามารถเข้าถึงแต่ละสมาชิกได้โดยการใช้ซับสคริบ (อาจจะบวกหรือลบ) ในกรณีของพอยน์เตอร์ไปยังอะเรย์ เมื่อมีการใช้ซับสคริบจะเป็นการกระโดดข้ามอะเรย์ทั้งชุด พอยน์เตอร์ไปยังอะเรย์มีประโยชน์เมื่อใช้กับอะเรย์หลายมิติ สำหรับการประกาศทำได้โดย "int (*ap) [N]" โดย N เป็นขนาดของอะเรย์ ถ้าไม่มี N จะไม่รู้ขนาดของอะเรย์ซึ่งไม่มีใครใช้กัน

6. เราจะ allocate สำหรับอะเรย์หลายมิติที่ปรับค่าได้อย่างไร

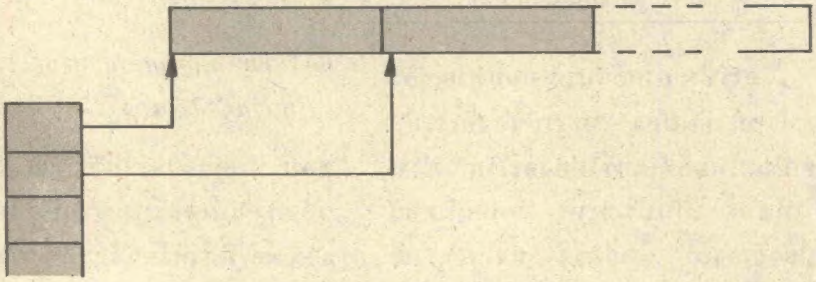
ตอบ ปกติแล้วจะใช้วิธี allocate อะเรย์ของพอยน์เตอร์ก่อน หลังจากนั้นจึง allocate ที่สำหรับแต่ละพอยน์เตอร์ วิธีนี้จะช่วยประหยัดความจำแต่จะได้พื้นที่

```
int **array = (int **) malloc(nrows * sizeof(int *));
for(i = 0; i < nrows; i++)
        array[i] = (int*)malloc(ncolumns*sizeof(int***));
```



รูปที่ 1

```
int **array = (int **)malloc(nrows * sizeof(int *));
array[0] = (int *)malloc(nrows * ncolumns * sizeof(int));
for(i = 1; i <nrows; i++)
        array[i] = array[0] + i * ncolumns;
```



รูปที่ 2

ไม่ต่อเนื่องแบบอะเรย์จริง ๆ ดังรูปที่ 1 เราสามารถทำให้พื้นที่ allocate ต่อเนื่องกันตลอดได้ดังรูปที่ 2

ในทั้งสองกรณีนี้ เราอ้างถึงสมาชิกที่ i, j ได้ โดยใช้รูปแบบของอะเรย์สองมิติคือ

```
array [i] [j]
```

ทั้งสองวิธีนั้นจะต้องอ่านค่าพอยน์เตอร์ถึงสองครั้งซึ่งอาจทำให้ช้าได้ ในกรณีนี้แก้ไขได้โดยจำลองอะเรย์สองมิติโดยใช้อะเรย์มิติเดียวดังนี้

```
int *array = (int *)malloc(nrows * ncolumns
                * sizeof(int));
```

อย่างไรก็ตาม การอ้างถึงสมาชิกเราต้องคำนวณเองดังนี้

```
array [i* ncolumns + j]
```

เป็นการอ้างถึงสมาชิก i, j (เราสามารถใช้มาโครเพื่อความสะดวกได้ แต่จะได้รูปแบบการเข้าถึงสมาชิกเป็นแบบไม่ใช่อะเรย์สองมิติ)

จากบทความทั้ง 2 ตอนที่กล่าวถึงภาษาซี คงจะทำให้ผู้อ่านได้เข้าใจถึงปัญหาต่าง ๆ ที่มักจะเกิดขึ้นและสามารถนำไปประยุกต์ใช้ในการเขียนโปรแกรมได้